YUI recommends YUI 3.

YUI 2 has been deprecated since 2011. This site acts as an archive for files and documentation.

Yahoo! UI Library

YAHOO Global 2.2.0

Modules

animation
autocomplete
button
calendar
connection
container
datasource
datatable
dom
dragdrop
element
event
history
logger
menu
slider
tabview
treeview
yahoo

Classes

YAHOO
YAHOO.env
YAHOO.lang
YAHOO_config

Files

YAHOO.js

```
/**
 * The YAHOO object is the single global object used by YUI Library.  It
 * contains utility function for setting up namespaces, inheritance, and
 * logging.  YAHOO.util, YAHOO.widget, and YAHOO.example are namespaces
 * created automatically for and used by the library.
 * @module yahoo
 * @title  YAHOO Global
 */

/**
 * YAHOO_config is not included part of the library.  Instead it is an object
 * that can be defined by the implementer immediately before including the
 * YUI library.  The properties included in this object will be used to
 * configure global properties needed as soon as the library begins to load.
 * @class YAHOO_config
 * @static
 */

/**
 * A reference to a function that will be executed every time a YAHOO module
 * is loaded.  As parameter, this function will receive the version
 * information for the module. See <a href="YAHOO.env.html#getVersion">
 * YAHOO.env.getVersion</a> for the description of the version data structure.
 * @property listener
 * @static
 */
if (typeof YAHOO == "undefined") {
    /**
     * The YAHOO global namespace object.  If YAHOO is already defined, the
     * existing YAHOO object will not be overwritten so that defined
     * namespaces are preserved.
     * @class YAHOO
     * @static
     */
    var YAHOO = {};
}

/**
 * Returns the namespace specified and creates it if it doesn't exist
 * <pre>
 * YAHOO.namespace("property.package");
 * YAHOO.namespace("YAHOO.property.package");
 * </pre>
 * Either of the above would create YAHOO.property, then
 * YAHOO.property.package
 *
 * Be careful when naming packages. Reserved words may work in some browsers
 * and not others. For instance, the following will fail in Safari:
 * <pre>
 * YAHOO.namespace("really.long.nested.namespace");
 * </pre>
 * This fails because "long" is a future reserved word in ECMAScript
 *
 * @method namespace
 * @static
 * @param  {String*} arguments 1-n namespaces to create
 * @return {Object}  A reference to the last namespace object created
 */
YAHOO.namespace = function() {
    var a=arguments, o=null, i, j, d;
    for (i=0; i<a.length; i=i+1) {
        d=a[i].split(".");
        o=YAHOO;

        // YAHOO is implied, so it is ignored if it is included
        for (j=(d[0] == "YAHOO") ? 1 : 0; j<d.length; j=j+1) {
            o[d[j]]=o[d[j]] || {};
            o=o[d[j]];
        }
    }

    return o;
};

/**
 * Uses YAHOO.widget.Logger to output a log message, if the widget is
 * available.
 *
```

```
 * @method log
 * @static
 * @param  {String}  msg  The message to log.
 * @param  {String}  cat  The log category for the message.  Default
 *                        categories are "info", "warn", "error", time".
 *                        Custom categories can be used as well. (opt)
 * @param  {String}  src  The source of the the message (opt)
 * @return {Boolean}      True if the log operation was successful.
 */
YAHOO.log = function(msg, cat, src) {
    var l=YAHOO.widget.Logger;
    if(l && l.log) {
        return l.log(msg, cat, src);
    } else {
        return false;
    }
};


/**
 * Initializes the global by creating the default namespaces and applying
 * any new configuration information that is detected.
 * @method init
 * @static
 */
YAHOO.init = function() {
    this.namespace("util", "widget", "example");
    if (typeof YAHOO_config != "undefined") {
        var l=YAHOO_config.listener,ls=YAHOO.env.listeners,unique=true,i;
        if (l) {
            // if YAHOO is loaded multiple times we need to check to see if
            // this is a new config object.  If it is, add the new component
            // load listener to the stack
            for (i=0;i<ls.length;i=i+1) {
                if (ls[i]==l) {
                    unique=false;
                    break;
                }
            }
            if (unique) {
                ls.push(l);
            }
        }
    }
};

/**
 * Registers a module with the YAHOO object
 * @method register
 * @static
 * @param {String}   name    the name of the module (event, slider, etc)
 * @param {Function} mainClass a reference to class in the module.  This
 *                             class will be tagged with the version info
 *                             so that it will be possible to identify the
 *                             version that is in use when multiple versions
 *                             have loaded
 * @param {Object}   data      metadata object for the module.  Currently it
 *                             is expected to contain a "version" property
 *                             and a "build" property at minimum.
 */
YAHOO.register = function(name, mainClass, data) {
    var mods = YAHOO.env.modules;
    if (!mods[name]) {
        mods[name] = { versions:[], builds:[] };
    }
    var m=mods[name],v=data.version,b=data.build,ls=YAHOO.env.listeners;
    m.name = name;
    m.version = v;
    m.build = b;
    m.versions.push(v);
    m.builds.push(b);
    m.mainClass = mainClass;
    // fire the module load listeners
    for (var i=0;i<ls.length;i=i+1) {
        ls[i](m);
    }
    // label the main class
    if (mainClass) {
        mainClass.VERSION = v;
        mainClass.BUILD = b;
    } else {
        YAHOO.log("mainClass is undefined for module " + name, "warn");
    }
};

/**
 * YAHOO.env is used to keep track of what is known about the YUI library and
 * the browsing environment
 * @class YAHOO.env
 * @type Object
 * @static
 */
YAHOO.env = YAHOO.env || {
    /**
     * Keeps the version info for all YUI modules that have reported themselves
     * @property modules
     * @type Object[]
     */
```

```
    modules: [],

    /**
     * List of functions that should be executed every time a YUI module
     * reports itself.
     * @property listeners
     * @type Function[]
     */
    listeners: [],

    /**
     * Returns the version data for the specified module:
     *      <dl>
     *      <dt>name:</dt>      <dd>The name of the module</dd>
     *      <dt>version:</dt>   <dd>The version in use</dd>
     *      <dt>build:</dt>     <dd>The build number in use</dd>
     *      <dt>versions:</dt>  <dd>All versions that were registered</dd>
     *      <dt>builds:</dt>    <dd>All builds that were registered.</dd>
     *      <dt>mainClass:</dt> <dd>An object that was was stamped with the
     *                 current version and build. If
     *                 mainClass.VERSION != version or mainClass.BUILD != build,
     *                 multiple versions of pieces of the library have been
     *                 loaded, potentially causing issues.</dd>
     *       </dl>
     *
     * @method getVersion
     * @static
     * @param {String}  name the name of the module (event, slider, etc)
     * @return {Object} The version info
     */
    getVersion: function(name) {
        return YAHOO.env.modules[name] || null;
    }
};

/**
 * Provides the language utilites and extensions used by the library
 * @class YAHOO.lang
 */
YAHOO.lang = {
    /**
     * Determines whether or not the provided object is an array
     * @method isArray
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isArray: function(obj) { // frames lose type, so test constructor string
        if (obj.constructor && obj.constructor.toString().indexOf('Array') > -1) {
            return true;
        } else {
            return YAHOO.lang.isObject(obj) && obj.constructor == Array;
        }
    },

    /**
     * Determines whether or not the provided object is a boolean
     * @method isBoolean
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isBoolean: function(obj) {
        return typeof obj == 'boolean';
    },

    /**
     * Determines whether or not the provided object is a function
     * @method isFunction
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isFunction: function(obj) {
        return typeof obj == 'function';
    },

    /**
     * Determines whether or not the provided object is null
     * @method isNull
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isNull: function(obj) {
        return obj === null;
    },

    /**
     * Determines whether or not the provided object is a legal number
     * @method isNumber
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isNumber: function(obj) {
        return typeof obj == 'number' && isFinite(obj);
    },

    /**
     * Determines whether or not the provided object is of type object
     * or function
     * @method isObject
```

```
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isObject: function(obj) {
        return typeof obj == 'object' || YAHOO.lang.isFunction(obj);
    },

    /**
     * Determines whether or not the provided object is a string
     * @method isString
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isString: function(obj) {
        return typeof obj == 'string';
    },

    /**
     * Determines whether or not the provided object is undefined
     * @method isUndefined
     * @param {any} obj The object being testing
     * @return Boolean
     */
    isUndefined: function(obj) {
        return typeof obj == 'undefined';
    },

    /**
     * Determines whether or not the property was added
     * to the object instance.  Returns false if the property is not present
     * in the object, or was inherited from the prototype.
     * This abstraction is provided to enable hasOwnProperty for Safari 1.3.x.
     * There is a discrepancy between YAHOO.lang.hasOwnProperty and
     * Object.prototype.hasOwnProperty when the property is a primitive added to
     * both the instance AND prototype with the same value:
     * <pre>
     * var A = function() {};
     * A.prototype.foo = 'foo';
     * var a = new A();
     * a.foo = 'foo';
     * alert(a.hasOwnProperty('foo')); // true
     * alert(YAHOO.lang.hasOwnProperty(a, 'foo')); // false when using fallback
     * </pre>
     * @method hasOwnProperty
     * @param {any} obj The object being testing
     * @return Boolean
     */
    hasOwnProperty: function(obj, prop) {
        if (Object.prototype.hasOwnProperty) {
            return obj.hasOwnProperty(prop);
        }

        return !YAHOO.lang.isUndefined(obj[prop]) &&
                obj.constructor.prototype[prop] !== obj[prop];
    },

    /**
     * Utility to set up the prototype, constructor and superclass properties to
     * support an inheritance strategy that can chain constructors and methods.
     *
     * @method extend
     * @static
     * @param {Function} subc   the object to modify
     * @param {Function} superc the object to inherit
     * @param {Object} overrides  additional properties/methods to add to the
     *                              subclass prototype.  These will override the
     *                              matching items obtained from the superclass
     *                              if present.
     */
    extend: function(subc, superc, overrides) {
        var F = function() {};
        F.prototype=superc.prototype;
        subc.prototype=new F();
        subc.prototype.constructor=subc;
        subc.superclass=superc.prototype;
        if (superc.prototype.constructor == Object.prototype.constructor) {
            superc.prototype.constructor=superc;
        }

        if (overrides) {
            for (var i in overrides) {
                subc.prototype[i]=overrides[i];
            }
        }
    },

    /**
     * Applies all prototype properties in the supplier to the receiver if the
     * receiver does not have these properties yet.  Optionally, one or more
     * methods/properties can be specified (as additional parameters).  This
     * option will overwrite the property if receiver has it already.
     *
     * @method augment
     * @static
     * @param {Function} r  the object to receive the augmentation
     * @param {Function} s  the object that supplies the properties to augment
     * @param {String*}  arguments zero or more properties methods to augment the
     *                             receiver with.  If none specified, everything
```

```
     *                            in the supplier will be used unless it would
     *                            overwrite an existing property in the receiver
     */
    augment: function(r, s) {
        var rp=r.prototype, sp=s.prototype, a=arguments, i, p;
        if (a[2]) {
            for (i=2; i<a.length; i=i+1) {
                rp[a[i]] = sp[a[i]];
            }
        } else {
            for (p in sp) {
                if (!rp[p]) {
                    rp[p] = sp[p];
                }
            }
        }
    }
};

YAHOO.init();

/*
 * An alias for <a href="YAHOO.lang.html">YAHOO.lang</a>
 * @class YAHOO.util.Lang
 */
YAHOO.util.Lang = YAHOO.lang;

/**
 * An alias for <a href="YAHOO.lang.html#augment">YAHOO.lang.augment</a>
 * @for YAHOO
 * @method augment
 * @static
 * @param {Function} r  the object to receive the augmentation
 * @param {Function} s  the object that supplies the properties to augment
 * @param {String*}  arguments zero or more properties methods to augment the
 *                             receiver with.  If none specified, everything
 *                             in the supplier will be used unless it would
 *                             overwrite an existing property in the receiver
 */
YAHOO.augment = YAHOO.lang.augment;

/**
 * An alias for <a href="YAHOO.lang.html#extend">YAHOO.lang.extend</a>
 * @method extend
 * @static
 * @param {Function} subc   the object to modify
 * @param {Function} superc the object to inherit
 * @param {Object} overrides  additional properties/methods to add to the
 *                              subclass prototype.  These will override the
 *                              matching items obtained from the superclass
 *                              if present.
 */
YAHOO.extend = YAHOO.lang.extend;
```

Copyright © 2007 Yahoo! Inc. All rights reserved.